Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Technology 3 (2012) 187 – 195

Procedia Technology

The 2012 Iberoamerican Conference on Electronics Engineering and Computer Science

An FPGA Implementation for Image Interpretation Based on Adaptive Boosting Algorithm in the Real-Time Systems

Mario-Alberto Ibarra-Manzano[a], Dora-Luz Almanza-Ojeda[b]

[a] *Digital Signal Processing Laboratory; Electronics Department; DICIS; University of Guanajuato*
Carretera Salamanca-Valle de Santiago Km.3.5+1.8 Km, Comunidad Palo Blanco, 36885, Salamanca, Guanajuato, Mexico
[b] *Departamento de Ingeniería Robótica; Universidad Politécnica de Guanajuato*
Av. Universidad Norte SN, Comunidad Juan Alonso, 38483, Cortazar, Guanajuato, Mexico

Abstract

This paper presents an FPGA architecture for objects classification based on Adaptive Boosting algorithm. The architecture uses the color and texture features as input attributes to discriminate the objects in a scene. Moreover, the architecture design takes into account the requirements of real-time processing. To this end, it was optimized for reusing the texture feature modules, giving, in this way, a more complete model for each object and becoming easier the object-discrimination process. The reuses technique allows to increase the information of the object model without decrease the performance or drastically increase the area used on the FPGA. The architecture classifies 30 dense images per second of size 640×480 pixels. Both, classification architecture and optimization technique, are described and compared with others architectures founded in the literature. The conclusions and perspectives are given at the end of this document.

Keywords: Adaptive Boosting Algorithm, Texture and Color Features, FPGA Architecture, Real-Time Systems

1. Introduction

One of the most used technique on the obstacles detection consist in modeling the objects based on appearance (color, texture,...) and/or geometric attributes. This model must deliver a features vector, by means of this vector it is possible to separate an image in different regions. The feature vector is used as input of the classification algorithm during the learning process. Furthermore, the classification algorithm requires a training set which is used to discriminate the classes by dividing the image in regions or objects in according to an attributes vector. In this case, the number of dimensions n depends on the number of attributes to be analyzed. It is important to points out that in this methodology, the algorithm to calculate the attributes plays an important role during the final classification, since the performance of detection is directly proportional to the information contained in the attributes vector. On the other hand, the classification algorithm is also important because it allows to obtain the best trace off among performance, training time and classification time.

Email addresses: ibarram@ugto.mx (Mario-Alberto Ibarra-Manzano), luzdora@ieee.org (Dora-Luz Almanza-Ojeda)

doi:10.1016/j.protcy.2012.03.020

Color and texture features are the most frequently used attributes for classifying objects in indoor scenes, because they allow to discriminate several objects. Also, color feature is an attribute that contains sufficient information about the objects, in addition, it does not require much computational power. However, if the environment contains several and different kind of objects, the the color feature is not enough, and it is necessary to complement our information by other feature such as texture. However, the texture attribute implies a disadvantage: texture attributes calculation requires a high computational cost. To overcome this inconvenient, two possible alternatives have been proposed : (1) make a sparse analysis and (2) use dedicated systems. First alternative consist in dividing the image into several regions in which only few points are analyzed to determine the class on the region, thus the calculation time is highly reduced and the real-time performance could be reached. Second alternative proposes to carry out the features analysis on a dedicated system such as a DSP, GPU or FPGA, with the aim of increasing the performance and, unlike first alternative, achieving a dense analysis of the image, that is pixel by pixel.

In the literature, there are different that uses texture analysis on dedicated systems. One of the first solutions to this issue employing an heterogeneous embedded system was proposed by Ibarra-Pico et al [1]. The system consist of a DSP and an FPGA and it is based on a primary variant of the algorithm of sums and differences of histograms (SDH) proposed by Unser [2]. In [1], the authors use as input the image for calculating the histograms, from such the texture attributes are obtained straightforward. However, these new attributes do not hold some characteristics unlike the SDH algorithm proposed by Unser, as a consequence, the performance during classification is reduced. A another solution based on the algorithm of concurrence matrices, was proposed by Tahir [3] in which the performance in classification was improved. The solution proposed consist in obtaining the texture attributes using an FPGA, since this computation requires more computing power. On the other hand, the classification process is carried out on a PC reducing the overall performance in system. In this case, the implementation on a PC cannot be avoided because most of the FPGA resources are already consumed by the texture analysis. This system is used for processing medical images in order to detect cancer. To overcome the problem of the system performance, the image is divided into several regions and each region is classified taking into account all the pixels. This image division allows to reduce the output image with respect to the input image, mainly due to the classification process is performed out to the feature analysis process.

In Lopez-Estrada et. al [4], the authors provide a solution to reduce the consumed area in the texture analysis. In this work, the architecture calculates the texture attributes as in [3] by taking the modules only for the texture features that consume the lowest quantity of resources on the FPGA. Thus, Lopez-Estrada could implement the classification process on an FPGA. However, as this done by a decision tree, the performance of the classification is limited. Furthermore, this strategy forces the entire image to have the ”predominant” class resulted, i.e, the system provides a class for each image but it ignores the presence of another classes and the position of such resulting class in the image. After in 2010, Ibarra-Manzano et al. [5, 6] present an adaptation of the SDH algorithm for texture feature analysis allowing to calculate dense attributes for each of the pixels in the image. In spite of a significant reduction in the arithmetic computation of the SDH algorithm, the architecture shows some modules not optimized, moreover most of them are not reused even if that could improve the global performance of the system. In order to overcome this problems, same authors propose in 2011 an extension of their work creating an optimal architecture for a greater amount of image information, from which it is possible to increase the number of attributes [7].

In this work, it is used an analysis optimized of textures for providing a most complete model of the objects and measurably improving the classification results. At this point, it is necessary to consider the case of distinguishing a particular object among several objects of the same kind but with different color. Usually, we found that this particular situation requires more than only texture features information. For this reason, a color model is included to the texture classification architecture with the aim of providing more information to discriminate singular objects. Once texture and color features have been analyzed and the corresponding attributes values found, it is necessary to select a classification method to increase the efficiency and performance of the global strategy. Therefore, we found the AdaBoost learning advanced techniques as the most convenient method due to it is easier to be implemented on an FPGA but essentially because it adaptively updates the distribution of samples by increasing the weights that were poorly represented by the previous classification. Furthermore, AdaBoost algorithm can be very effective with high-dimensional data

since it manages multiple classifiers called "weak" for obtaining more accurate results [8]. In this case, the decision tree technique is used as "weak" classifiers. In the next section, we provide an overview about our final architecture based on the Adaptive Boosting (AdaBoost) classification algorithm and the technique of decision tree. Next, the architecture that implements the AdaBoost algorithm is presented in section 3. The performance results and a comparative analysis of similar architectures found in the literature are presented in Section 4. Conclusions and perspectives are presented at the end of this document.

2. Boosting Algorithm

The AdaBoost algorithm was developed in 1995 by Freund and Schapire [9] and it is considered a powerful technique which provides a solution to the classification problem using supervised learning. The AdaBoost combines the performance of several classifiers called "weak" in order to obtain a strong classifier. Freund and Schapire have proved that if a weak classifier is slightly better than chance, the error of the final classifier decreases exponentially. Each weak classifier is usually simple and requires a short time of computing. Several weak classifiers efficiently combined produce a strong classifier that, commonly, outperforms the "monolithic" fort classifiers such as SVM and neural networks.The objective of the algorithm consist in finding the minimal function in the hypothesis space, also this function must gives the smallest error in the distribution D. This distribution is issued of the analysis of samples in the training set. The AdaBoost algorithm shows two main differences compared to other machine learning techniques:

1. Adapt the errors in the hypothesis given by the weak classifiers
2. The limit of the operation depends only of the weak classifiers model that is generated by the distribution function D

Moreover, the algorithm takes as inputs a set of training samples $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$ where each x_i belongs to space X and each label y_i is an element of the set Y. In this work, we assume that $Y = \{-1, +1\}$ represents the negative and positive class, respectively. The algorithm AdaBoost employs a weak classifier as base that is continuously used in series of $t = 1, ..., T$ iterations. It is important to points out that the distribution D must be provided during the training process as well as a set of weights on all the training samples. The weight of the distribution function, for the training sample i at iteration t is referred to as $d_t(i)$. Initially, all weights are equal and uniform (see Equation 1), but at each iteration, the weights of misclassified training samples are increased. Therefore, the next weak classifier is particularly focused on samples difficult to classify, and vice versa when the training samples are classified.

$$D_1(i) = \frac{1}{m} \quad \forall i \tag{1}$$

On the other hand, the learning process of weak classifiers trains on m samples in order to find the best weak hypothesis $h_t : X \rightarrow \{-1, +1\}$ for the weight distribution D_t. In addition, learning process is repeated T times on the same set of samples, however at each iteration the distribution D_t is updated and, as a consequence, the hypothesis found at each iteration will change also. The weak hypothesis can be seen as one attributes that allow us to distinguish between two classes, with an error expressed by $\epsilon_t > 1/2$. According to this, the error is calculated as the sum of sample weights which were misclassified. Once the weak hypothesis h_t has been chosen, the AdaBoost algorithm intuitively calculates a parameter α_t, that is used to measure the importance assigned to weak hypothesis h_t. The final hypothesis H (see Equation 2) is constructed as a combination of weak hypotheses found at each iteration. The final hypothesis produced by AdaBoost algorithm has errors of $\epsilon 1, \epsilon 2, ..., \epsilon_T$, where the error ϵ_t for iteration t is defined by the equation 3. The error can be updated, yielding:

- The coefficient α_t of the weak hypothesis h_t.
- The new weights of the training samples (D_t)

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{2}$$

$$\epsilon_t = Pr_{D_t}\left[h_t(x_i) \neq y_i\right] \tag{3}$$

Algorithm 1 Adaptive Boosting

Require: $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$, o $x_i \in X \wedge y_i \in Y = \{-1, +1\}$
  $D_1(i) \Leftarrow \frac{1}{m}$
  for $t = 1, 2, ..., T$ **do**
    $h_t : X \rightarrow \{-1, +1\}$ {Learning weak hypothesis on the distribution D_t}
    $\epsilon_t \Leftarrow Pr_{D_t}\left[h_t(x_i) \neq y_i\right] = \sum_i D_t(i)\ \forall h_t(x_i) \neq y_i$ {Calculate the error learning}
    $h_t \Leftarrow h_t | min(\epsilon_t)$ {Find the weak hypothesis h_t with minimal learning error ϵ_t and distribution D_t.}
    $\beta_t \Leftarrow \frac{\epsilon_t}{1-\epsilon_t}$
    $\alpha_t \Leftarrow -\frac{1}{2} ln(\beta_t)$ {Calculate the coefficient of the weak hypothesis h_t}
    $D_{t+1}(i) \Leftarrow \frac{D_t(i)}{Z_t} \times exp(-\alpha_t y_i h_t(x_i))$ {Calculate the distribution function D_{t+1}, where z_t is the normalization factor}
  end for
  $H(x) \Leftarrow sign\left(\sum_t \alpha_t h_t(x)\right)$ {Obtain the final hypothesis}

The learning process of AdaBoost classifier is presented in Algorithm 1. In the classification phase, a new sample under test requires the final hypothesis, which combines T learned weak hypotheses from the training set. All weak hypotheses $h_1, h_2, ..., h_T$ carries out the classification of this sample in parallel, the result returned by h_i is affected by the corresponding weight α_i. A linear combination of these results provides the probability of sample belongs to the class. This probability is positive when the sample belongs to the class and negative otherwise. This result will provide the class, so that, we will assign to the sample, the class value expressed as to $+1$ or -1. This method can be seen as a majority vote in which, the class that has the most votes will be chosen for the sample under test.

Furthermore, the AdaBoost algorithm is easy to implement. It only requires one parameter for adjusting the number of weak hypotheses T, which directly depends on the threshold error of we want to achieve. As no special requirements are imposed for weak hypothesis, so AdaBoost can be combined with any learning method. However, we note that the AdaBoost algorithm is sensitive to noise producing negative effects: the algorithm could focused on several samples difficult to classify. According to this, it will exist a zone of ambiguity inter classes on the hypothesis space, affecting the performance of the final classifier. Therefore, we must be focused on the training set used during learning process.

In the AdaBoost algorithm, we chose the decision trees method as weak classifier especially Classification and Regression Trees (CART). This method has the advantage of being easy to implement on hardware for classification, which justifies this choice for the final implementation of a reprogrammable platform. The performance given by this method is sufficient to be combined with the AdaBoost algorithm. We describe the classification tree method in more detail in the next section.

2.1. Decision Trees

Once AdaBoost algorithm have been described, in this subsection, we explain the essential characteristics of the decision trees used as weak classifiers. A decision tree is a predictive model used during the classification and regression tasks. A decision tree is called classification tree when it classifies an object when it uses the values of their attributes in order to assign it to a class.

The single point of departure in a classification tree is called *root node* and consists of all training samples m at the top of the tree. A *node* is a subset of all variables, and it can be a leaf node or non-terminal. We will very often consider simple decision trees with only one root node per tree because they provide enough information and at the same time it avoids to carried out unnecessary computations.

A classification tree can be seen as an ordered sequence of questions in which followed questions formulated at each stage of the classification process depend on the answers to the previous questions. The sequence ends with a prediction of the class.

3. Hardware Implementation

The architecture for classification based on color and texture attributes was developed for an FPGA device that is a general engine for high-speed image processing. In order to maximize the performance, our design was made up to minimize resources (memory, Logic Elements) required on the FPGA. Our architecture is composed of three main operations, namely, the color transformation(for details, see [6]), the calculus of texture features(for details, see [7] or [5]) and the classification of color-texture features. Figure 1 shows, inside of a doted rectangle, the integration of the object classification modules for one camera. The hardware implementation of classification module is detailed in following subsections.

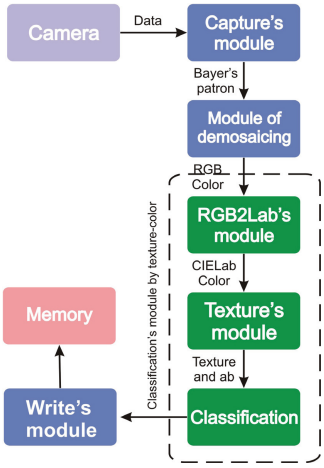

Fig. 1. Diagram classification algorithm based on color and texture attributes.

3.1. Weak classifier module

As we introduced above, the decision trees technique is used as the basis for the module of the weak hypothesis. A depth of $n_t = 5$ is used in each decision tree: we set this value in all decision trees to reduce the complexity of higher-level modules in the classification architecture, and, at the same time, to define a single module reconfigurable for all the decision trees. Figure 2 shows a part of the decision tree. This module has as input the data bus x of size $n_f \times n_b$ bits, where n_f represents the number of attributes coded on n_b bits. The number of attributes is $6 \times n_d + 2$, where n_d represents the number of displacements in the attributes of texture, 6 is the number of attributes of texture per displacements, and 2 the number of attributes of color obtained from the transformation CIE-Lab. The hardware architecture of this module consists of 1 level of selection and 6 levels of comparisons. The selection level consists in $2^{n_t+1} = 63$ multiplexers, controlled by the bus J, that choose the attributes on the bus x to compare these attributes in each particular branch of the decision tree. Each level of comparison, except the first, consists of a multiplex stage and a comparison stage. The multiplexing stage is used to select the attribute and the threshold that must be compared. The multiplexers are controlled by the results of previous levels of comparison, as this determines the branch of the tree where the current pixel is defined by its attributes on the bus x. The first 5 comparison levels correspond to levels of n_t depth of the decision tree, the last level of comparison is the output of the decision tree that is different for each leaf of the tree. The thresholds C are controlled by the bus of the same name, at the same time, controlled by the higher level module. The latency of this module is $2 \times (n_t + 1) = 12$ pulses of the pixel clock.

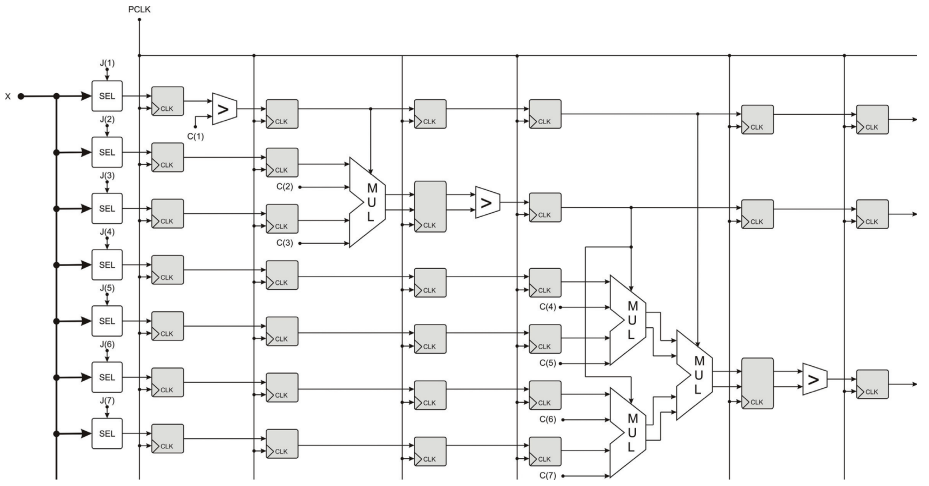

Fig. 2. The architecture of the weak classifier module that containts a decision tree.

3.2. AdaBoost Module

Figure 3 shows a part of the architecture of the classification module based on the Adaptive Boosting algorithm which consists of T weak hypothesis modules. AdaBoost module configures the bus: J for the attributes in the selectors and C for the thresholds in the multiplexers. For each of the modules of the weak hypothesis, this amounts to configure $2^{n_t+2} = 126$ values automatically to reduce errors. The final decision of each weak hypothesis is multiplied by the weight α corresponding to weight that is generated automatically and wired from the learning method implemented. The weight α represents a kind of "confidence" assigned to each weak hypothesis, and multiplication is a weighting of the final decision of each tree. The results of the weights, ie the outputs of the multipliers are added by using a pyramidal architecture to obtain the final classification, which is the sum of final decisions of all modules of weak hypotheses. The latency of this module is $2 \times (n_t + 1) + \lceil \log_2 (T) \rceil + 1$ pulses of the pixel clock.

4. Performance discussion and Results

We evaluated our classification architecture based on the attributes of color and texture. It was carried out on a FPGA EP3C120F780C7N Cyclone III family device of Altera [10]. The results of the synthesis are summarized in the table 1. The resources required are mainly based on the size of the image, the number of displacements, the size of the processing window used by the analysis module of the attributes of texture and the number of weak hypothesis used by the classification module. In this architecture, we use a image size of 640×480 pixels with only 1 displacement on the textures attributes of pseudo-variance, variance and correlation and 10 displacements on the others texture attributes (mean, contrast and homogeneity) and a processing windows size of 17×15 pixels. Finally, we compare the performance and resource consumption to 10, 20 and 30 weak hypothesis on the classification module. With these parameters and a clock rate of 50 MHz, our architecture classified 30 images par second.

In table 1, the first column shows the different types of resources consumed by the module. The second column shows the statistics of the resources consumed by the processing module of color. Because the coefficients in the transformation matrix are constant, this module uses only logic elements. Unlike the processing module of the color, the texture analysis module uses memory to store the pixels needed to calculate the texture attributes using the adequacy of sum and difference histograms. We have significantly reduced memory resources due to the fact that the processing windows used are common to different attributes and our architecture use more efficiently all the texture modules. An efficient reuses of the modules lets to reduce the computation resources moreover to improve the quantity of information. This module uses the embedded multipliers to calculate the attributes of contrast and variance. The other three columns in the table 1 refer

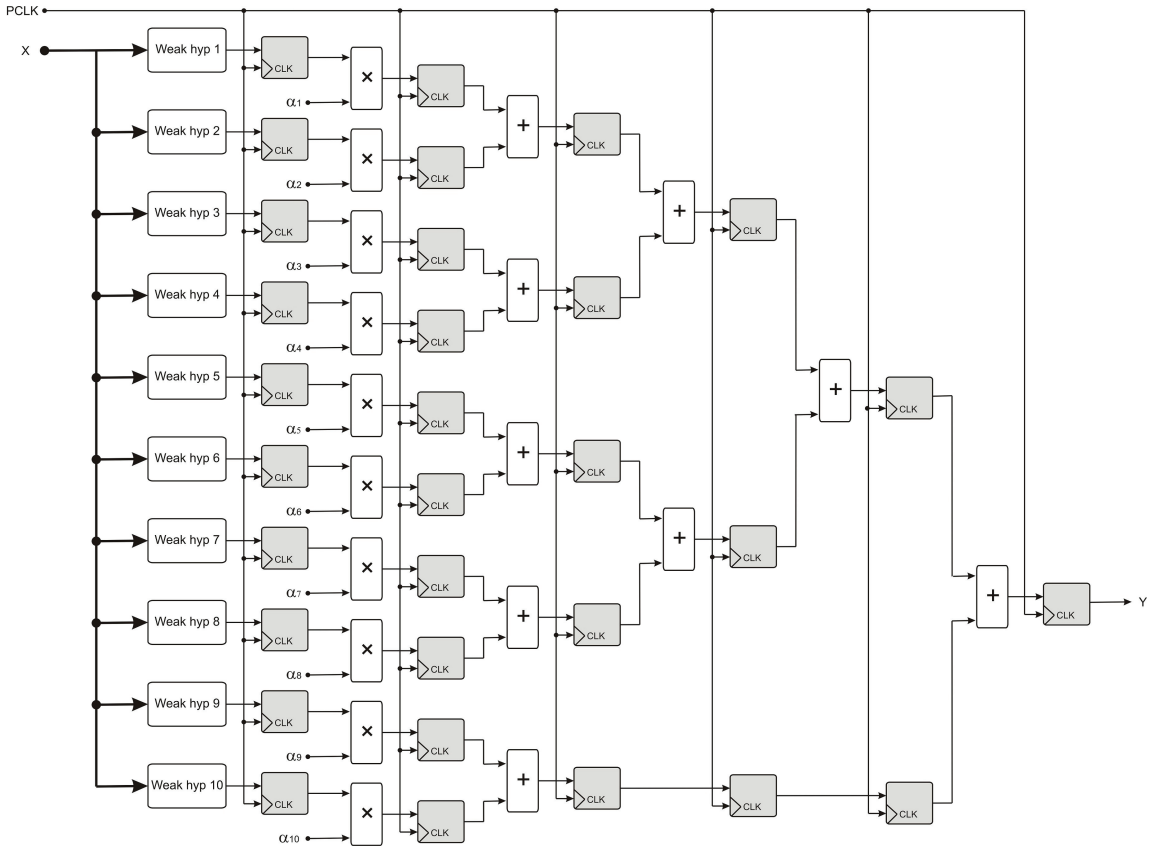

Fig. 3. The architecture of the classification module based on the AdaBoost algorithm.

to classification module based on the AdaBoost algorithm for a number of 10, 20 and 30 weak hypothesis. The number of logic elements is directly proportional to the number of weak hypotheses. Conversely, the number of embedded multipliers is obtained by subtracting the number of weak hypothesis (total weight) to the number of weights that are multiples of two, and this is that in this case, the multiplier is replaced by a shift operation.

Table 1. Table of statistics of resources consumed by the classification architecture.

	Attributs of color	Attributs of texture	Classification (# weak hypothesis) 10	20	30
Logic Elements	716	19,766	7,683	14,078	19,401
Combinational functions	643	15,676	5,676	11,615	16,826
Dedicated registres	280	12,591	5,229	8,210	10,543
Memory size	–	1.75 Mb	–	–	–
Embedded multipliers	–	120	36	68	64

We compare the performance between the same algorithms implemented on hardware on a FPGA, and software on a PC MacBook Pro with an Intel Core 2 Duo 2.33 GHz, 4 Mb L2 cache and 2 Gb of RAM. The results of the performance between the two solutions are presented in table 2 for different image sizes and number of weak hypothesis (the first and second columns, respectively). The third column shows the processing time for each implementation, the difference in units of time is noticeable. In the case of FPGAs,

the parallel processing and flexibility of computing can achieve processing times lower. The last column gives a speedup factor compared to the PC solution. It may be noted that this factor is directly proportional to the number of weak hypotheses. These results also emphasize that the number of weak hypothesis does not affect the processing time in the FPGA, because they are calculated simultaneously in contrast to the PC solution for which the duration of processing is directly related to their number. But unlike the FPGA, the PC solution is easy and quick to implement.

Table 2. Comparative table of performance for different implementations.

Image size (pixels)	# of weak hypothesis	Processing time		Speedup factor
		PC	FPGA	
320×240	30	$617ms$	$7.68ms$	80.37
640×480	10	$1.52s$	$30.72ms$	49.66
640×480	20	$1.96s$	$30.72ms$	63.83
640×480	30	$2.45s$	$30.72ms$	79.96
640×480	60	$3.80s$	$30.72ms$	123.9
1024×768	30	$6.43s$	$78.64ms$	81.76

Finally, the classification architecture proposed on FPGA has been compared to others three architectures found in the literature. Table 3 shows the results of the implementation of our architecture and the other three architectures. Our architecture and the previous version of this without reuse resources. Our architecture uses both the attributes of color and texture, the others use only the attributes of texture. Our architecture and the architecture of Lopez-Estrada [4] use the classification on the FPGA, which increases performance. However, the decision tree used in the architecture of Lopez-Estrada is a classifier less robust because it uses a smaller number of attributes, which reduces the scope of its use. The surface, in our architecture, can not be directly compared to architectures Tahir [3] and Lopez-Estrada which are implemented on FPGAs from Xilinx. However, it is important to note that unlike our architecture, the other two deal with the image so that there is no overlap between the processing windows, ie they do not one class for each pixel in the image, but give a class for each set of pixels in the processing window. This reduces the number of operations and therefore the resources and increases performance. In fact, the architecture of Lopez-Estrada gives one class for the entire image, ie, it detects whether the class exists in the picture or not but it does not indicate what part of the this is the object, it detects only the existence.

Table 3. Comparative table of the performance of classification algorithms based on the appearance attributes for different FPGA architecture.

	Our Design	Ibarra-Manzano [6]	Tahir [3]	Lopez-Estrada [4]
Color features	CIE-L*ab	CIE-L*ab	–	–
Texture features	ASDH	ASDH	GLCM	GLCM
Classification algorithm	AdaBoost	AdaBoost	LOO (PC)	Decision Tree
Image size	640×480	640×480	512×512	256×256
Size of processing window	17×15	17×15	128×128	256×256
Gray level	256	256	32	256
Number of attributes	35	8	7	3
Performance	30 fps	30 fps	26 fps	–
Surface	34, 560 LE	34, 560 LE	20, 544 S	277 S
Size of internal memory	1.75 Mb	379 Kb	400 Kb	–
Size of external memory	–	–	8 Mb	–

5. Conclusions and Perpectives

This paper describes how a complex classification function has been implemented on an FPGA-based architecture. It has been proven that same results are obtained with a software and the FPGA-based implementation. This effort agreed to design, implement and evaluate this architecture is justified for different reasons. By now the 30Hz frequency is a bound due to the camera characteristics; we could improve the reactivity with faster acquisitions. Using only software on a single core, it is not possible by now, to compute at 30 Hz AdaBoost algorithm on 640×480 images, even with an optimized code.

Future works will be devoted to the evaluation of high level synthesis tool to generate such complex architectures. In order to improve the classification performance, an enriched attributes module consisting of more features is been analyzed. The total parameterization of the modules in function of the performance and resources will help to accelerate the design-time. At the same time the design will have a more accurate estimation resources and will let known an a priori performance value during the classification process.

The outlook with respect to the embedded implementation are the method implementation in a Xilinx FPGA, this in order to compare the performance achieved with a different FPGA technology. In the same way, we want to make a comparison with others embedded technologies as DSP or GPU, however, in the last case highlighting not only the performance but also the power consumption. To complement the comparative analysis, we will make an analysis of energy consumption in both: for the final architecture and in each module, in order to complement the information of classification module.

Acknowledgment

This work was partially funded by the CONACyT with the project entitled “Diseño y optimización de una arquitectura para la clasificación de objetos en tiempo real por color y textura basada en FPGA”, too we would like to express our gratitude for the Universidad de Guanajuato with the project entitled “Sistema de Identificación de objetos mediante los atributos del color y la textura utilizando arquitectura reconfigurable con FPGA” with number 000175/11.

References

[1] F. Ibarra Pico, S. Cuenca Asensi, V. Corcoles, Accelerating statistical texture analysis with an fpga-dsp hybrid architecture, in: FCCM ’01: Proceedings of the the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, IEEE Computer Society, Washington, DC, USA, 2001, pp. 289–290. doi:http://dx.doi.org/10.1109/FCCM.2001.7.

[2] M. Unser, Sum and difference histograms for texture classification, IEEE Trans. Pattern Anal. Mach. Intell. 8 (1) (1986) 118–125. doi:http://dx.doi.org/10.1109/TPAMI.1986.4767760.

[3] M. A. Tahir, A. Bouridane, F. Kurugollu, An fpga based coprocessor for glcm and haralick texture features and their application in prostate cancer classification, Analog Integr. Circuits Signal Process. 43 (2) (2005) 205–215. doi:http://dx.doi.org/10.1007/s10470-005-6793-2.

[4] S. Lopez-Estrada, R. Cumplido, Decision tree based fpga-architecture for texture sea state classification, in: Reconfigurable Computing and FPGA’s, 2006. ReConFig 2006. IEEE International Conference on, 2006, pp. 1–7. doi:10.1109/RECONF.2006.307770.

[5] M. Ibarra-Manzano, D.-L. Almanza-Ojeda, J.-M. Lopez-Hernandez, Design and optimization of real-time texture analysis using sum and difference histograms implemented on an fpga, in: Electronics, Robotics and Automotive Mechanics, 2010. CERMA 2010. Conference on, 2010, pp. 325–330.

[6] M. Ibarra-Manzano, M. Devy, J.-L. Boizard, Real-time classification based on color and texture attributes on an fpga-based architecture, in: A. Morawiec, J. Hinderscheit (Eds.), Proceedings of the 2010 Conference on Design and Architecture for Signal and Image Processing, Electronic Chips and Systems design Initiative, 2010, pp. 53–60.

[7] M.-A. Ibarra-Manzano, D.-L. Almanza-Ojeda, An fpga implementation for texture analysis considering the real-time requirements of vision-based systems, in: A. Koch, R. Krishnamurthy, J. McAllister, R. Woods, T. El-Ghazawi (Eds.), Reconfigurable Computing: Architectures, Tools and Applications, Vol. 6578 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2011, pp. 110–117.

[8] V. Athitsos, S. Sclaroff, Boosting nearest neighbor classifiers for multiclass recognition, in: Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on, 2005, p. 45. doi:10.1109/CVPR.2005.424. URL 10.1109/CVPR.2005.424

[9] Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application, Journal of Computer and System Sciences 55 (1) (1997) 119–139.

[10] A. C. http://www.altera.com/ [online] (March 2010). [link].